CHART2 NTCIP Driver High Level Design

High Level Design Study for the Design, Development and Integration of the NTCIP Driver into the CHART2 System

Edwards and Kelcey Technical Design Document for CHART2.

A study of the High Level Design for the NTCIP Driver to integrate into the CHART2 software system. The CHART2 software provides a series of interfaces for a DMS type Driver to interact with. The NTCIP is a set of standards developed by several interested bodies for Intelligent Trafficking Systems. This document determines the code architecture for an NTCIP DMS Driver to be added to the CHART2 software.

Author	Cameron Riley
Email	criley@ekmail.com
Company	Edwards and Kelcey Technology, Inc
URL	
Address	750 Miller Drive, Suite F-1
City	Leesburg, Virginia
Zip	20175
phone	703.779.7988
fax	703.779.7989
date	7th December 2001

Abstract

NTCIP is a grouping of standards for the meaningful communication between devies for ITS. The CHART2 system is the Maryland State Highway Administration's highway incident management program. This project is for the development and integration of an NTCIP driver for NTCIP compliant DMS's into the CHART2 software.

The CHART2 software provides strong interfaces for the integration of an NTCIP DMS into the CHART2 system. The central interface is the ProtocolHdlr interface which determines the communication and interaction between the CHART2 DMS object and the Message Sign.

NTCIP defines the communication at the Data Link layer as PMPP or the Point to Multi Point Protocol. The operations for encoding and decoding the PMPP frames are seperated into a seperate package. The NTCIPProtocolHdlr when requested to communicate with the sign, uses the PMPP package to create a PMPP frame and and when a response is received to decode the frame.

The PMPP protocol defines the information field as consisting of IPI and an SNMP PDU or alternatively an STMP PDU. The STMP protocol is not finalized and the SNMP is a more common internet protocol. Three SNMP libraries were evaluated by Edwards and Kelcey Technology, with joeSNMP being decided as the most suitable for CHART2's requirements.

The NTCIP standard defines the Mib groups for NTCIP compliance. These contain the Signs status in it's controller. The NTCIP driver uses MibXML to store the status of the CHART2 DMS locally in the CHART2 system. The MibXML is DMS instance specific. When requests from the CHART2 system are made against the NTCIPProtocolHdlr interface, the status of the NTCIPDMS will be maintained by the manipulation of the MibXML in the instance of the CHART2 NTCIPDMS.

The CHART2 DataPort interface is the session manager for opening direct communication with a Sign. The interface is a master only and doesnt support unsolicited asynchronous Trap requests.

CHART

CHART is the highway incident management program of the Maryland State Highway Administration. CHART is comprised of four major components, traffic monitoring, incident response, traveler information and traffic management. The traveler information component of CHART provides real-time information concerning travel conditions on main roads in the primary coverage area. The information is conveyed to travelers via either Variable Message Signs, Highway Advisory Radio, Commercial radio and television broadcasts and a telephone advisory service. The variable message signs are programmable message boards, both permanent and portable which are capable of displaying real-time traffic conditions to motorists. NTCIP compliant Dynamic Message Signs are a type of VMS. CHART2 is the current implementation of the CHART system being developed by the Maryland State Highways Administration.

CHART2 CORBA Object Transport Mechanism

The CHART2 System uses CORBA, to transport and communicate between objects across the distributed client and server system. The GUI's from which the DMS Devices are managed, exist as Clients in the CORBA system. The FMSServer carries the implementations of the Objects. The Driver interacts across this system by the IDL interfaces.

The ORB is a message bus between objects that may reside on any machine in the network. The IDL is the interfaces by which the distributed objects publish their capabilities. CORBA is the specification that describes the ORB's functionality. CORBA specifies location transparency and language transparency. For CHART2's distributed system, the former is the most important.

NTCIP Standard

The National Transportation Communications for Intelligent Transportation System Protocol (NTCIP) is a family of standards maintained by NEMA, AASHTO and ITE. The NTCIP standards provide the rules and vocabulary for electronic traffic equipment from different manufacturers to communicate and operate with each other. NTCIP compliant devices must follow this standard. For a Device to be NTCIP compliant it must implement a mandatory set of MIB's, accept data transport by PMPP and be capable of reading Multi Message Format for sign display.

Major Components of the NTCIP Driver

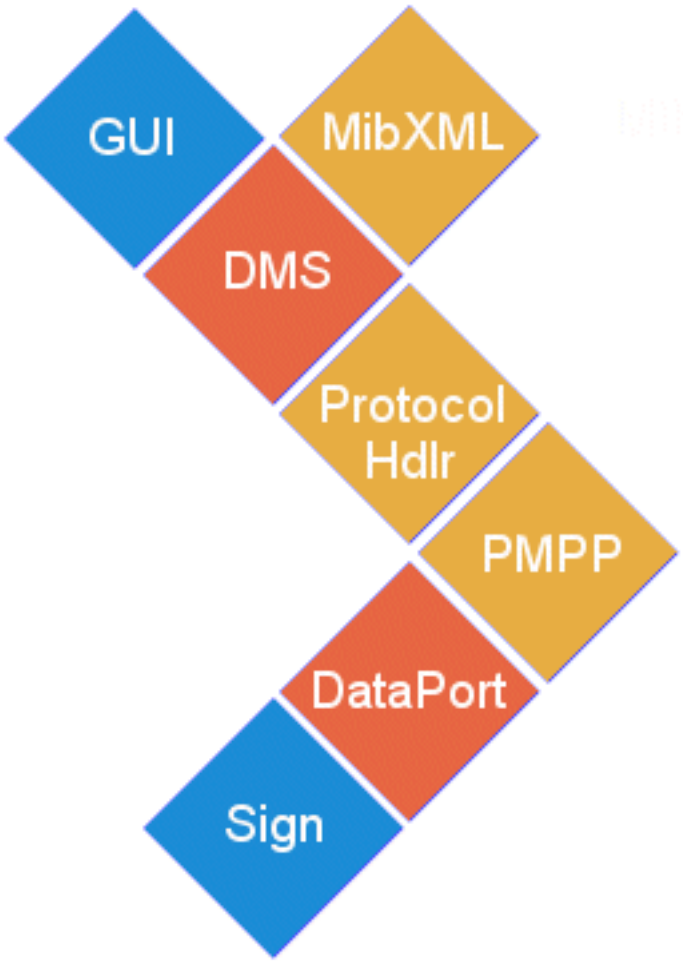


Diagram showing the showing major components in the NTCIP Driver. The GUI is the DMS GUI for an NTCIP specific sign, the DMS is the CHART2 DMS object, the MibXML is the NTCIP Device's status as represented by it's interaction with the Mib package. The Protocol Handler is the protocol handler interface which is provided by the CHART2 system. The Protocol Handler carries a set of actions for CHART2 DMS's that the DMS can access. To communicate with the sign, the PMPP package is used to encode and decode the packets. The CHART2 DataPort interface is the session manager, which encapsulates the physical connection to the sign.

Overview of Components

The major components of the NTCIP Driver for the CHART2 system are the CHART2 DMS object, the MibXML state storage mechanism, the Protocol Handler, the Pmpp component and the DataPort.

The DMS is the object which represents a Sign in the CHART2 software system. The DMS knows how to describe itself and maintains it's internal state and status. The DMS can use the Protocol Handler to communicate with the physical sign and query, or update it's status including the message. This is commonly done through the GUI interface.

The MibXML component is the Mib points and groups placed into a DMS instance specific XML structure in memory. The MibXML maintains the DMS's internal state including being updated with the responses to the DMS's requests to the physical Sign.

The NTCIP Protocol Handler is the interface provided by CHART2 for the actions that a CHART2 DMS can request. The Protocol Handler is the intermediary between the means of communication and the CHART2 DMS, it provides the DataPort, which is able to connect to a physical DMS.

The PMPP component is the means for transportation of data meaningfully between the CHART2 software system and the physical Sign's onboard controller, including the Management Information Base. The PMPP protocol follows the NTCIP set of standards for communication with trafficing devices.

The DataPort is the CHART2 interface for connecting across a physical line to a physical sign.

NTCIP Driver

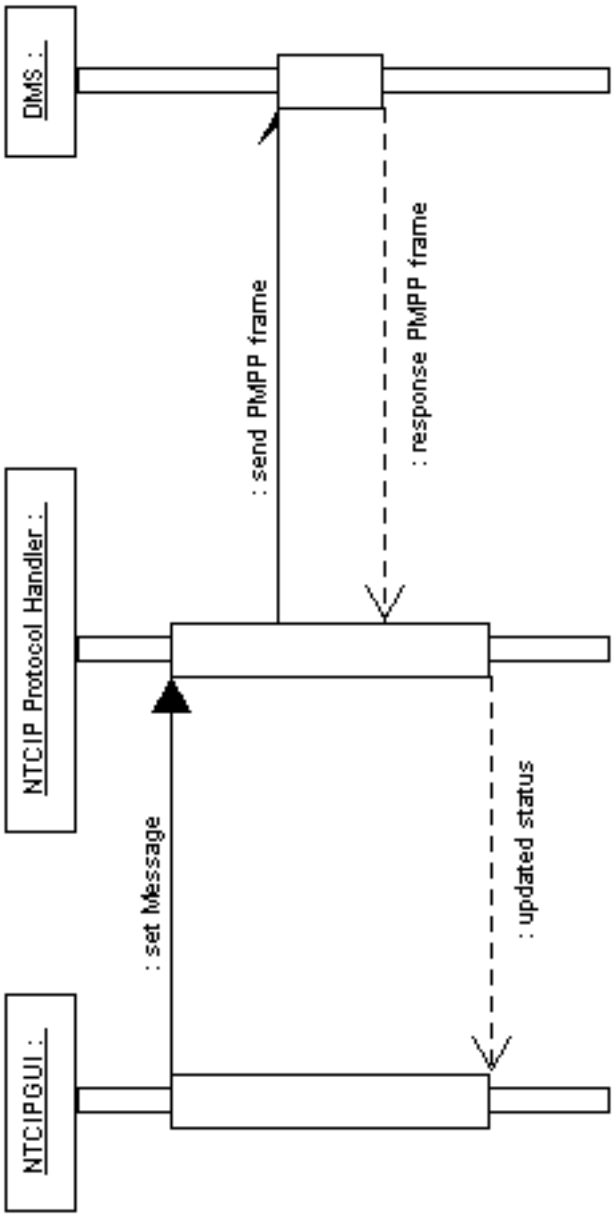
The software solution to the addition of the NTCIP Driver to the CHART2 system allows for the software to be divided into small packages which focus on particular tasks. The central component of the driver is the Protocol Handler. This is represented for Message Signs through the DMSProtocolHdlr Interface. An NTCIP specific Protocol Handler will implement this interface. The Protocol Handler has the following methods.

```
public void setConfiguration(DMSProtocolHdlrConfig config)
 throws DMSProtocolHandlerException;

public DMSProtocolHdlrConfig getConfiguration();


public void setMessage(DataPort port,
                       String multiMsg,
                       boolean beaconState)
  throws DMSProtocolHandlerException;

public void blank(DataPort port)
  throws DMSProtocolHandlerException;

public DMSDeviceStatus getStatus(DataPort port)
  throws DMSProtocolHandlerException;

public void reset(DataPort port)
  throws DMSProtocolHandlerException;
```

The setConfiguration() method sets the configuration parameters which are mostly used for

determining formatting for messages to signs. The get Configuration() method is an assessor method for returning the configuration object. The setMessage() method is for displaying a message on a Sign. The blank() method blanks the sign removing any currently displayed message. The getStatus() method returns the current status of the sign. The reset() method resets the Sign's controller.

The CHART2 system makes requests on these methods and the Protocol Handler creates suitable PMPP requests which update the DMS's current state, which is then reflected back into the CHART2 system.



Sequence Diagram describing the interaction between the GUI, Protocol Handler and the Sign.

PMPP Communication

The NTCIP set of standards defines the Point to Multi Point Protocol for communication with trafficing devices. PMPP is a specialization of the HDLC protocol which can use SNMP or STMP for the INFO field. The STMP standard is still being defined and the SNMP is already a widely accepted protocol on the internet for remote devices. The NTCIP driver will support SNMP through PMPP.

The PMPP standard places restrictions on the fields that make a up a frame or packet sequence. The frames must be bordered by the bit 01111110 or the hex 0x7E, the frame must support CRC 16 bit checksumming, bit stuffing and transparency, as well as BER encoded SNMP requests and responses.

PMPP frame structure. The PMPP frame is similar to an HDLC frame, with the major change being the addition of the IPI field in the HDLC information field.

The Protocol Handler interface and the CHART2 system expose the DataPort as the communication session interface. The DataPort IDL interface supports the following methods;

```
public void send(byte[] data)
 throws DataPortIOException;

public byte[] receive(long initialTimeoutMillis,
                      long interCharTimeoutMillis,
                      long maxReadDurationMillis)
   raises (DataPortIOException);
```

Where the send() method sends the binary over the port and the receive() method receives bursts of bytes from the port as the data chunks become available. The time the port remains listening can be specified through the initialTimeoutMillis which is the length of time to wait until the first byte of data is received, the interCharTimeoutMillis which is the amount of time to wait between two consecutive bytes. Once this times out it is assumed that the complete packet has been received. The final argument is the maxReadDurationMillis which is the maximum amount of time that should be spent receiving bytes after the initial byte is received.

The PMPP standard defines Traps for SNMP. The DataPort is a master that opens a port and sends binary data and then waits to receive a reply. Traps are asynchronous, unsolicited calls from the Sign. The trapping mechanism requires that the system be capable of fielding unsolicited PMPP calls from the Sign to the CHART2 system. The DataPort structure doesn't support this type of communication.

As SNMP is a widely used and adopted communication standard there are several SNMP libraries available for use, both commercial and opensource. Edwards and Kelcey Technology evaluated three closely, the Adventnet SNMP API, Outback Inc SNMP and joeSNMP.

All three follow the design of opening an SNMP Session and then communicating with SNMP Requests and Responses. This is the role in CHART2 which is satisfied by the DataPort. As the DataPort uses byte arrays to send and receive, the SNMP toolkit which will work with the greatest ease would require that the PDU be easily converted to and from a byte array. In this area joeSNMP

had the richest API.

The licensing fee's for Adventnets SNMP API were $11,500 per developer seat plus runtime costs which is prohibitive. The Outback Inc's SNMP toolkit was $995 for developer seat and deployment. Both however are commercial closed source products and offer no source code to the developer. joeSNMP is opensource and licensed under the LGPL library. There are no developer or runtime costs involved, however as it is opensource, there is no immediate support that comes with it's usage. In this area Edward and Kelcey Technology can be contracted to support the library for code maintenance, bug removal and addition of features.

The PMPP package will be separate to the Protocol Handler and placed in the CHART2.Utility.Communications package. The package will have an object which follows the Facade pattern and provides a simple enclosing interface to the PMPP package. It is through this facade object that the CHART2 system should access the packages capabilities. The facade object, PmppBuilder, carries four methods;

```
public byte[] encodeGet(byte[] address, OID oid)

public byte[] encodeSet(byte[] address, OID oid, String value)

public String decodeGet(byte[] buffer)
  throws InvalidFrameException

public void decodeSet(byte[] buffer)
  throws InvalidFrameException
```

Where the encodeGet() method creates a binary PMPP frame to get the value of the OID, the encodeSet() method creates a binary PMPP frame to set the value of the OID in the sign, the decodeGet() method unmarshalls the value from the received binary chunk and the decodeSet() method checks the frame from the received binary chunk. The address binary comes from the CHART2DMS's getId() method. Through these four methods the requirements for interaction with the NTCIP compliant DMS as defined in the Protocol Handler interface can be achieved.

The working object in the Communications package is the PmppRequest object. This object is tasked with storing the frames state, determining it's validity to the standard, as well as marshalling and unmarshalling the frame to an HDLC encoded frame and decoding the frame to it's constituent parts. The PmppRequest follows the PmppSyntax interface which contains the methods;

```
public int encodePmpp(byte[] buf,
                      int offset,
                      HdlcEncoder encoder);

public int decodePmpp(byte[] buf,
                      int offset,
                      HdlcEncoder encoder)
  throws InvalidFrameException;
```

Where the encodePmpp() method accepts a binary array, and encodes the bytes contained within to an HDLC encoding from the offset onwards. The returned int is the last encoded byte in the buffer. The decodePmpp() method decodes the HDLC encoding in the buffer from the offset onwards. The returned int is the offset of the last decoded byte. The decodePmpp method throws an InvalidFrameException if the buffer fails to be decoded due to frame errors. The HdlcEncoder object is analogous to the AsnEnoder and BerEncoder objects found in the joeSNMP library.

The HdlcEncoder object is aware of the requirements for transparency that the HDLC standard

places on PMPP frames. Transparency requires that the 0x7E's and 0x7D's in the frame be escaped with 0x7D 0x5E and 0x7D 0x5D respectively. The frame flags which are 0x7E are not included in the transparency.

The PMPP standard requires that a CRC-16 checksum be calculated and embedded into the FCS field. The FCS check during encoding is done after the SNMP Request, Address and Control fields have been added to the PMPP. For unmarshalling this is done after the PMPP packet is de-framed. The FCS check is called from the PmppRequests encodePmpp() and decodePmpp() methods. The working object for the CRC check is the CRC16 object. This object follows the java.util.zip.Checksum interface. The standard Sun java libraries include a CRC32 object, but no published CRC16 object. There is in the sun.misc package a CRC16 object, but it is undocumented and only appears with Sun JVM's. It also does not follow the Checksum interface. The Checksum interface has the methods;

```
public long getValue();

public void reset();

public update(int b);

public void update(byte[] b, int offset, int length);
```

Where the checksum is calculated through the addition of byes by the update methods and getValue() is the value of the checksum's value which can be obtained at any time. The concrete class for CRC16 adds the two publicly available methods;

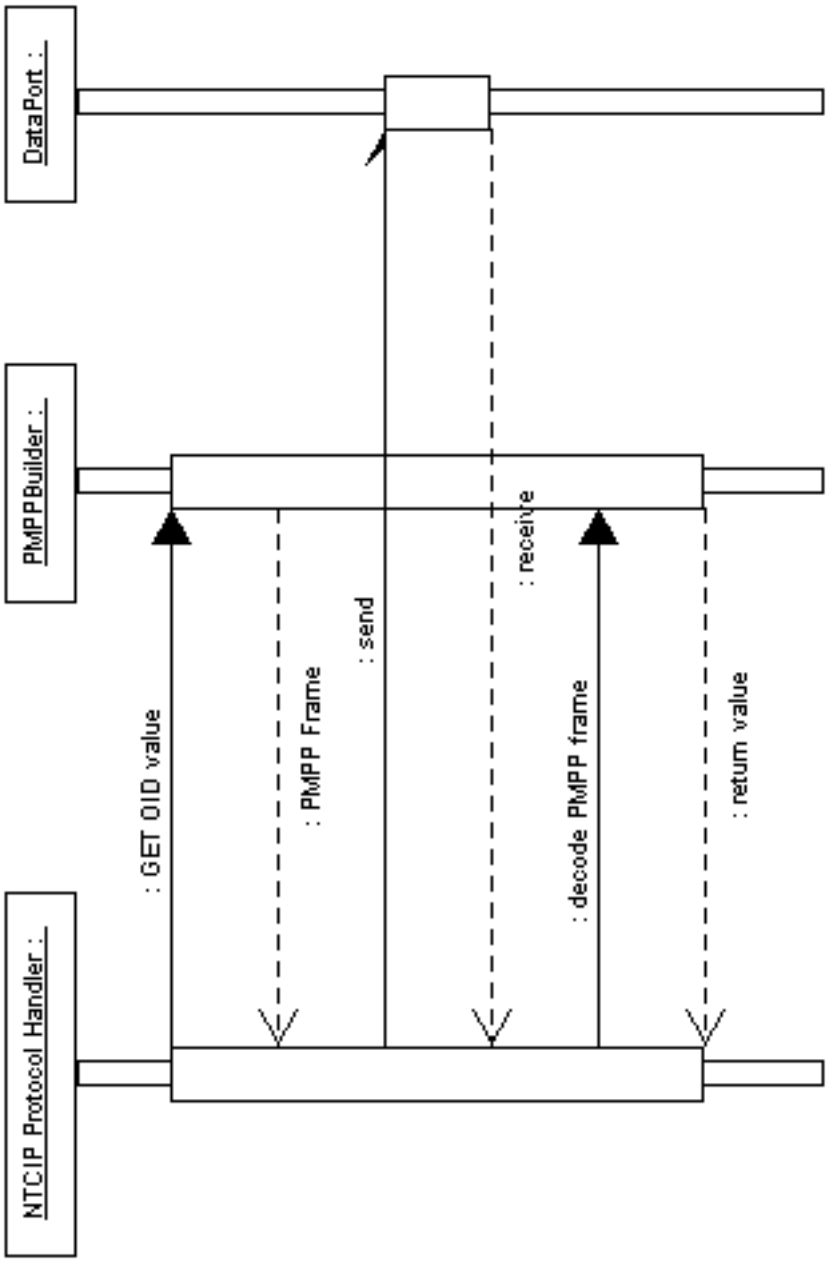
```
public int getHighByte()

public int getLowByte()
```

Which facilitates the checksum value being added quickly to the PmppRequest as the 16 bit field the PMPP standard requires.

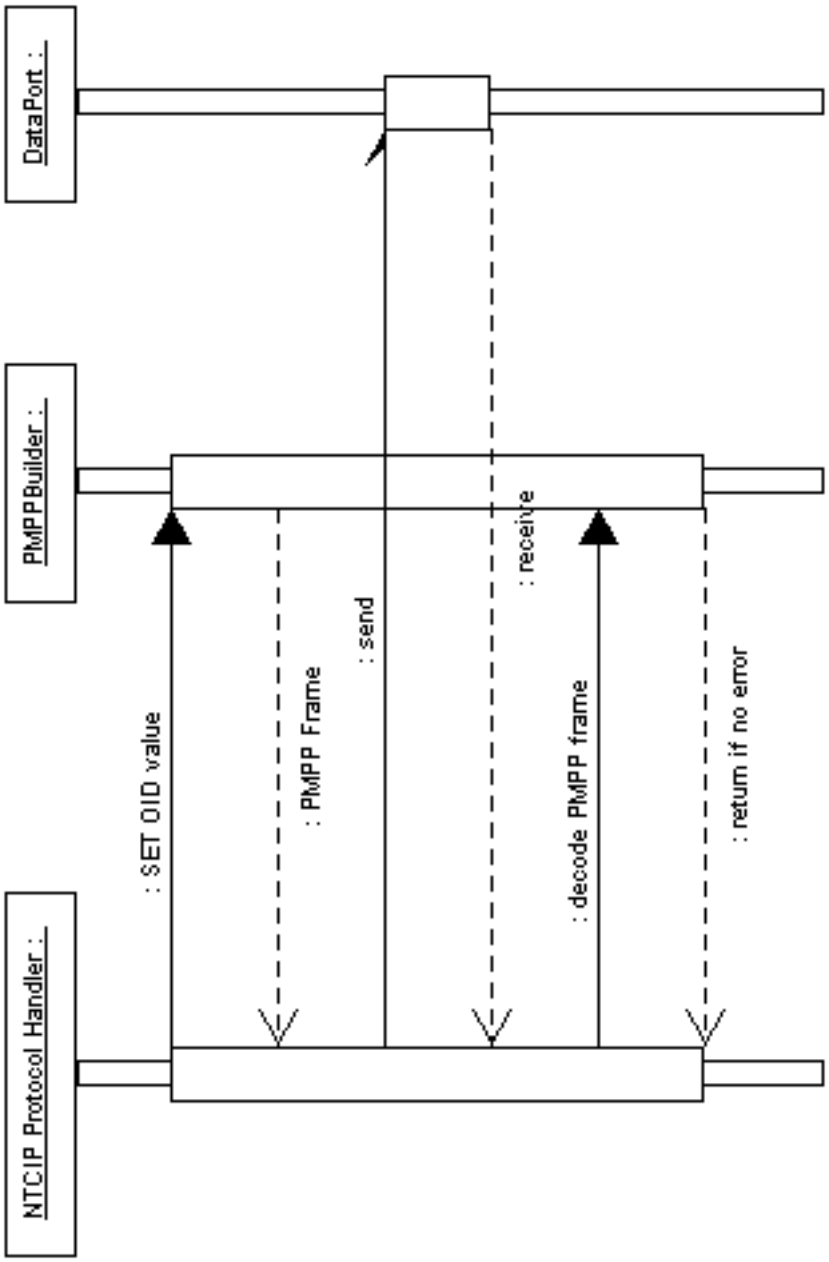
The SnmpPdu object is the CHART2 facade class to the joeSNMP library. The SnmpPdu object accepts through it's constructors the standard GET, SET and RESPONSE requests. The constructors are;

```
public SnmpPdu(OID oid, String value)
 throws UnsupportedEncodingException

public SnmpPdu(OID oid)
 throws UnsupportedEncodingException

public SnmpPdu(byte[] data)
 throws UnsupportedEncodingException
```

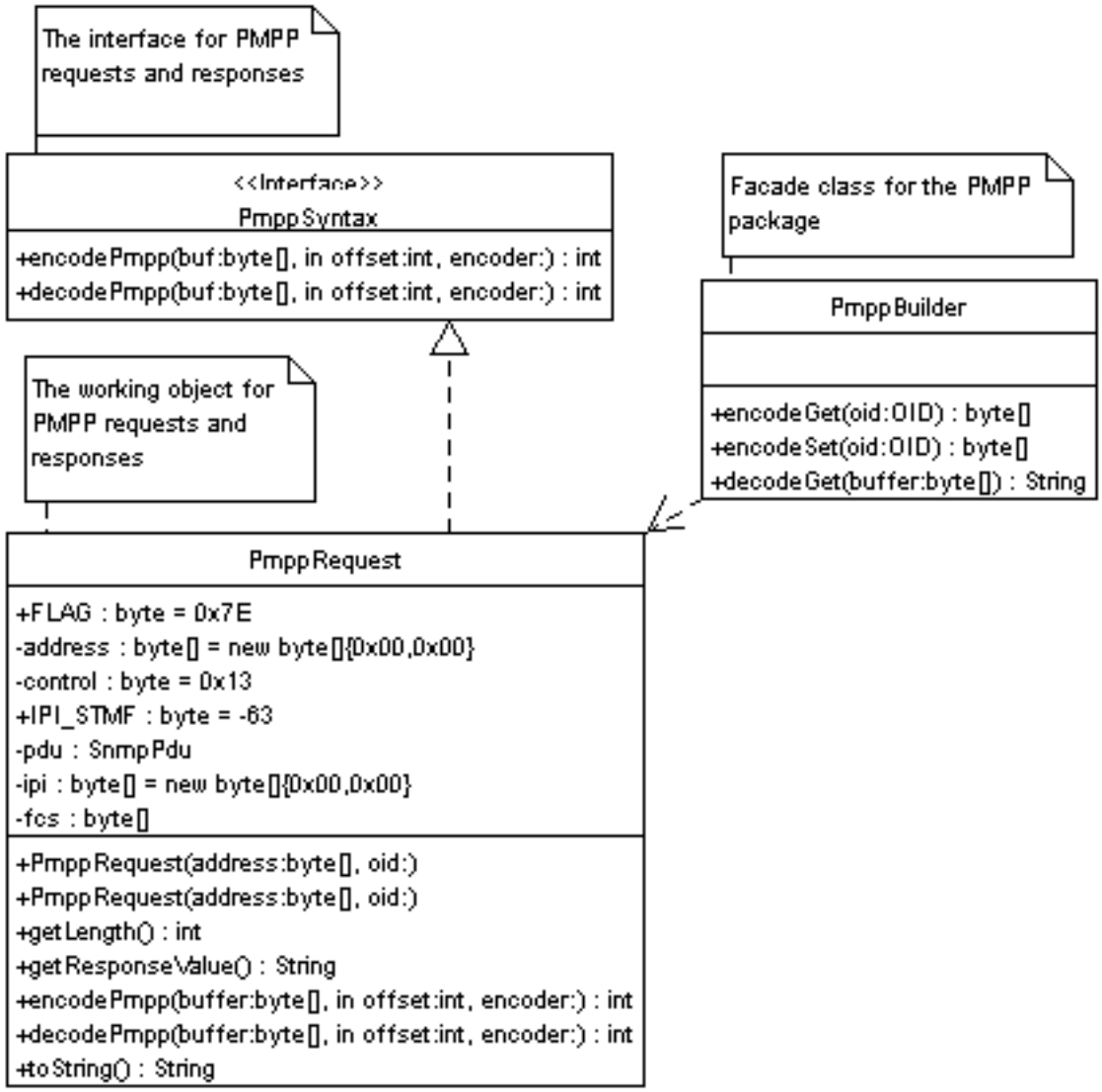
Where the first constructor which accepts an OID and value as argument is the constructor for an SNMP SET request the constructor which accepts an OID is an SNMP GET request and the constructor which accepts a binary array as an argument is the SNMP RESPONSE constructor. The SnmpPdu also contains encodeAsn() and decodeAsn() methods which are for marshalling and unmarshalling SNMP requests to and from BER Encoding.



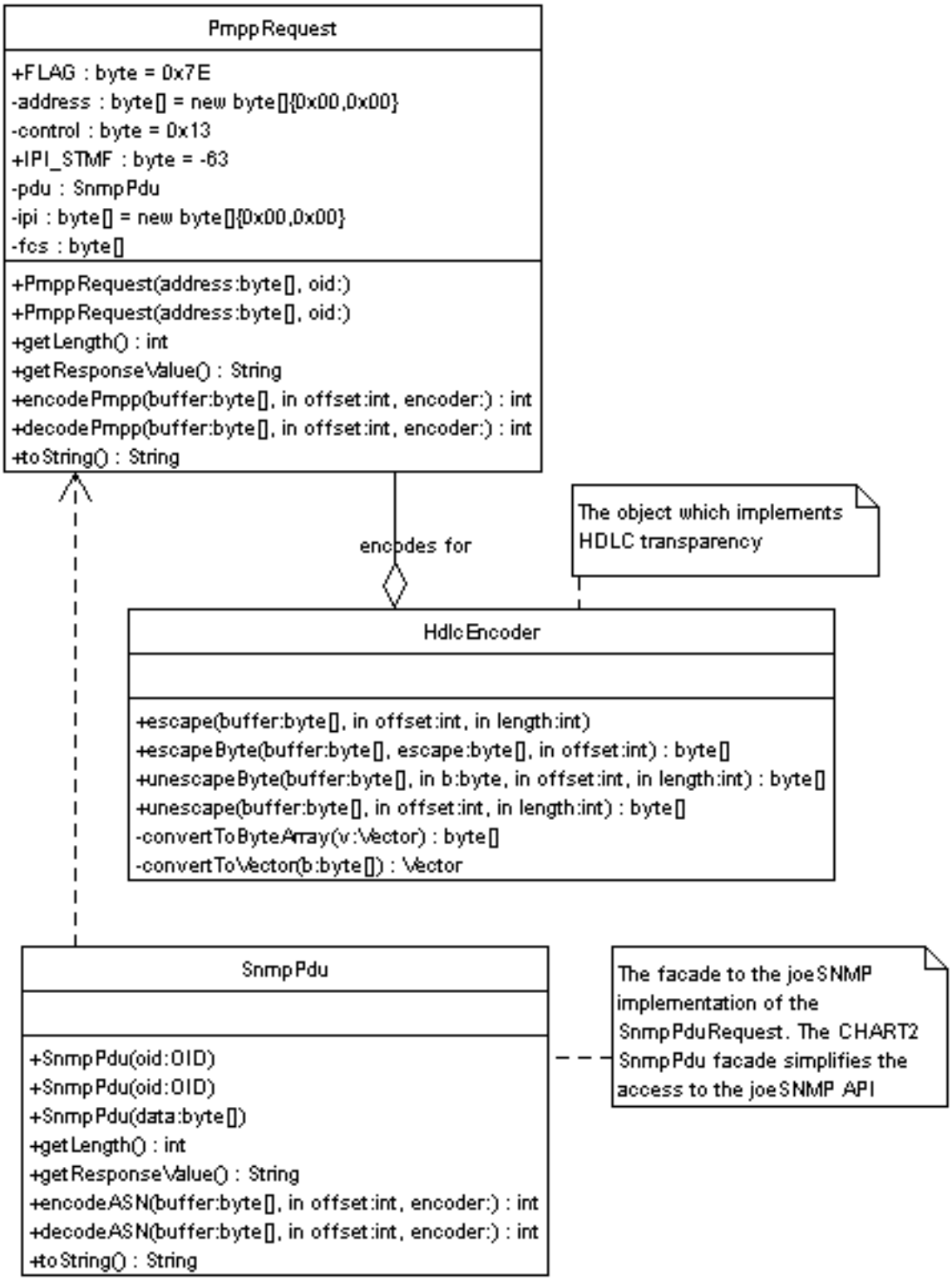
Sequence Diagram for a GET request describing the interaction between the NTCIP Protocol Handler, the PmppBuiler and the DataPort.



Sequence Diagram for a SET request describing the interaction between the NTCIP Protocol Handler, the PmppBuiler and the DataPort.



UML Diagram for the PmppSytnax interface, the PmppRequest working object and the PmppBuilder facade.



UML Diagram for the PmppRequest working object, the HdlcEncoder and the SnmpPdu facade.

MibXML : DMS State Storage

When the Protocol Handler acts upon requests from the Manager operating the CHART2 GUI, the Protocol Handler makes requests with the PMPP Protocol to the Sign and receives responses which reflect the DMS's current state. The Sign's state is stored in the actual device as MIB's or Management Information Base. A highly nodular standard which contains information on the OID, variable name, variable value and other descriptive fields describing the Mib point. The Mib's follow the ASN.1 standard which is commonly stored in an aged flat file format.

The NTCIP set of standards supports the Mib series, TS-3.2, TS-3.3, TS-3.5, TS-3.6 and TS-3.7, of which some groups are mandatory and others are optional. Not all the Mib's are required to support the Protocol Handler interface. The mandatory Mib groups include;

Mandatory Mib's under the NTCIP Specification;

- Configuration Conformance Group
- Security Node Conformance Group
- Sign Configuration and Capability Conformance Group
- Font Definition Conformance Group
- DMS Configuration Conformance Group
- Multiconfiguration Conformance Group
- Message Table Conformance Group
- Sign Control Conformance Group
- Illumination/Brightness Conformance Group
- Scheduling Conformance Group
- Sign Status Conformance Group

To achieve the status of the DMS being stored in the CHART2 system while maintaining separation of the Model(Mib), View(DMSGUI) and Controller(DMS) would be best stored in the DMS Object as an instance specific in memory Mib database using XML as the structure. This allows for the knowledge of the Mib's to be segregated from the Java code describing the DMS and the Protocol Handler. Other benefits include the ASN.1 nodular structure being stored in a well supported data structure with many libraries to store, search and manipulate the nodes, coarser information of the Mib point such as OID, name and description. The original Mib structure can be loaded once and cloned for instance specific DMS states.

The Mib objects are contained in the CHART2.Utility.Mib package, which exists in the CHART2 system to create, copy, store, request, insert and manipulate Mib's providing an accessible and simple interface to manage a CHART2 DMS's current state.

The Facade object for the Mib package is the MibService object. The MibService exposes three publicly available static methods;

```
public static List getGroupNames(MibManager mibmg)
public static OID getOID(MibManager mibmg,
```

```
                          String type)
  throws OIDNotFoundException, OIDNotValidException,
         JaxenException, SAXPathException

public static void setOIDValue(MibManager mibmg,
                               OID oid,
                               String value)
```

The method getGroupNames() gets a listing of the Groups the object with the MibManager interface contains. The method, getOID() returns an OID object from the object adhering to the MibManager interface and with the mib-name or mib-type by ASN.1 parlance. The method setOIDValue() set's the value of the OID. The MibManager interface contains methods to interact with a Mib database. The interface follows;

```
public static List getGroupNames()

public static OID getOID(String type)
  throws OIDNotFoundException, OIDNotValidException,
         JaxenException, SAXPathException

public static void setOIDValue(OID oid,
                               String value)
```

Which is similar to the methods exposed by the facade, however the interface is for objects which carry knowledge of the structure they are querying and manipulating.

The Mibdb object is the working object which contains the XML structure of the ASN.1 standard for Mib's as XML. It follows the MibManager interface and is the DMS instance specific object which contains the DMS's state. The Mibdb does nothing more than carry it's XML database and query against it. The MibServive serves mainly as a facade to the more complicated Mibdb working object.

The OID object represents a Mib point and contains the publicly accessible methods;

```
public String getName()

public String getOID()

public String getType()

public String getSystem()

public String getSyntax()

public String getSize()

public String getAccess()

public String getStatus()

public String getDescription()

public String getValue()
```

Which follows the nodes that the ASN.1 standard contains for a Mib point.

The Mibdb is instance specific to the DMS which is instantiated in the CHART2 system by an operator at a management station. The Mibdb is loaded as an empty XML structure from a server by the MibPool object. The MibPool follows the Singleton pattern and exists a single static instance on the server. The MibPool loads the XML which represents the Mib points and groups for an NTCIP compliant DMS from persistent storage when the CHART2 system is started. Persistent storage can be either from the file system or from the database. As it is only loaded once, the overhead for loading

the initial XML into the system is only felt the first time. When an NTCIP CHART2 DMS is instantiated in the CHART2 system, the DMS requests an empty Mibdb. The Mibdb is cloned from the XML template that the MibPool is carrying. The MibPool has the public methods;

```
public static MibPool getInstance()

public synchronized Mibdb getPool()
```

Where the getInstance() method is the initial manner in which to instantiate the MibPool on the JVM as the MibPool constructor is private. The synchronized method getPool() is the method to request the initial Mibdb for NTCIP compliant Signs.

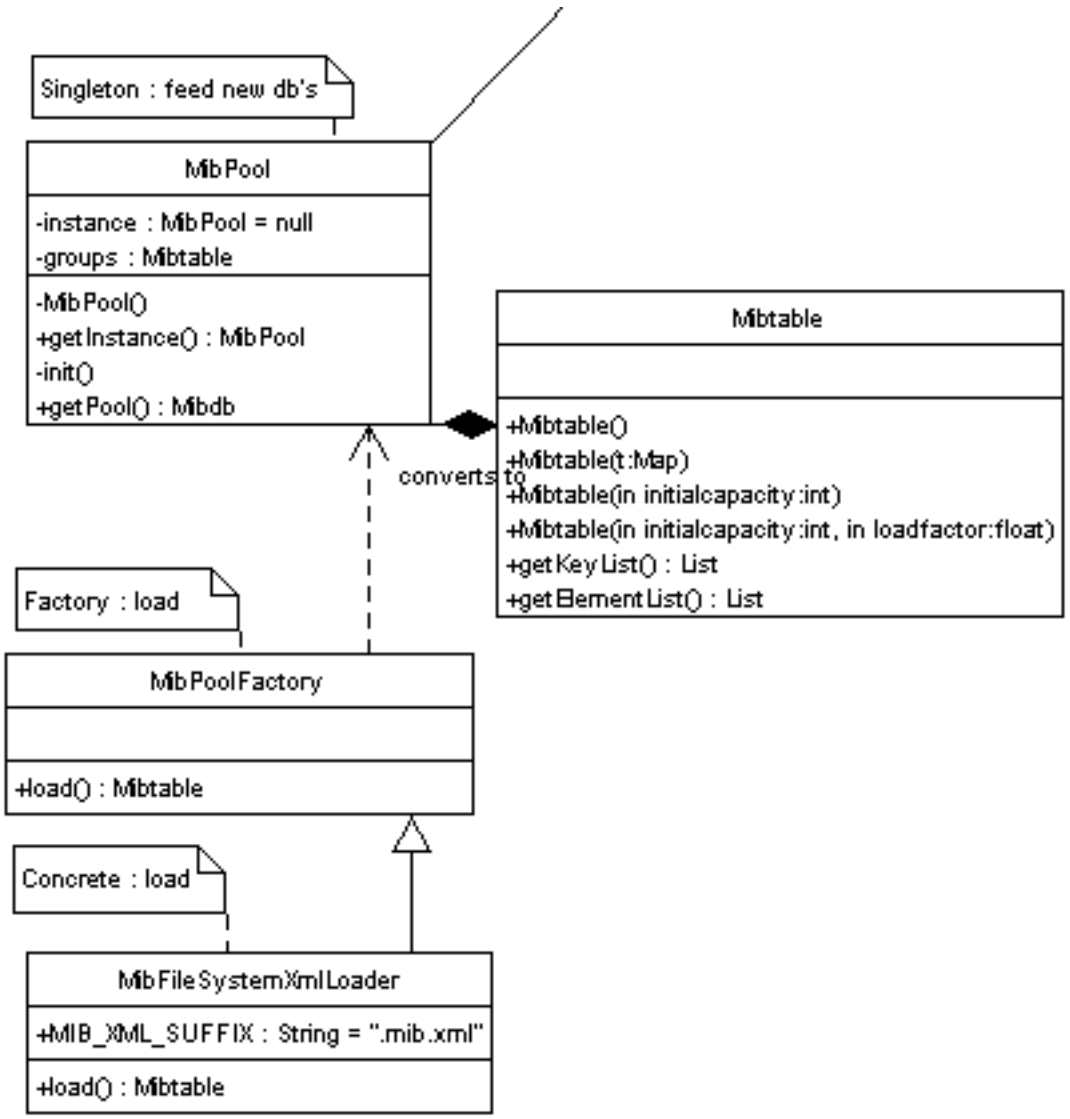
The Mib groups are fairly unchanging, and as such can be represented in an XML Document Type definition which describes the structure the MibXML has to follow. This is the template for the manner in which the data in the Mibdb must be structured.

```
<!ELEMENT root ANY>
<!ELEMENT name (#PCDATA) EMPTY>
<!ELEMENT definitions (#PCDATA) EMPTY>

<!ELEMENT group (description?,mib*,entry*) EMPTY>
<!ATTLIST group
  name CDATA #REQUIRED
  oid CDATA #REQUIRED
  system CDATA #REQUIRED
>

<!ELEMENT mib (description?) EMPTY>
<!ATTLIST mib
  oid CDATA #REQUIRED
  system CDATA #REQUIRED
  type CDATA #REQUIRED
  syntax CDATA #REQUIRED
  size CDATA #IMPLIED
  access CDATA #REQUIRED
  status CDATA #REQUIRED
>

<!ELEMENT description (#PCDATA) EMPTY>

<!ELEMENT entry (description?,mib*) EMPTY>
<!ATTLIST entry
  oid CDATA #REQUIRED
  system CDATA #REQUIRED
  type CDATA #REQUIRED
  syntax CDATA #REQUIRED
  size CDATA #IMPLIED
  access CDATA #REQUIRED
  status CDATA #REQUIRED
  index CDATA #REQUIRED
>
```

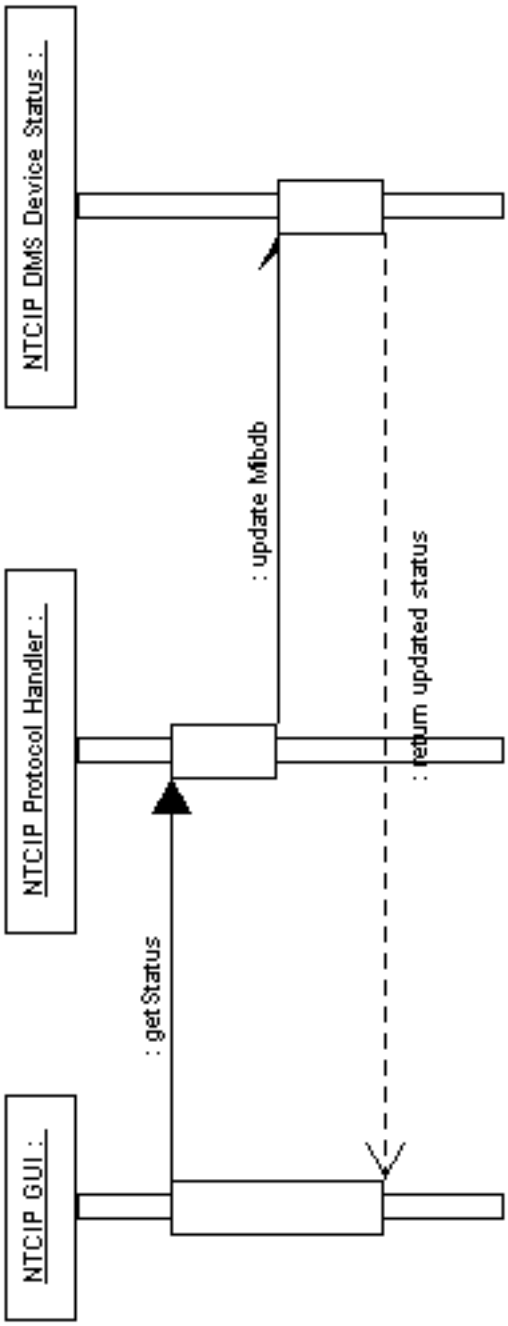
The DTD only guarantees that the structure be maintained in the flatfile representation of the XML and hence when loaded. The same DTD can be mapped into a Java object named MibDTD for the Mibdb to take advantage of;

```
/** DTD node value */
public static final String NODE_NAME = "name";

/** DTD node value */
public static final String NODE_DESCRIPTION = "description";
```

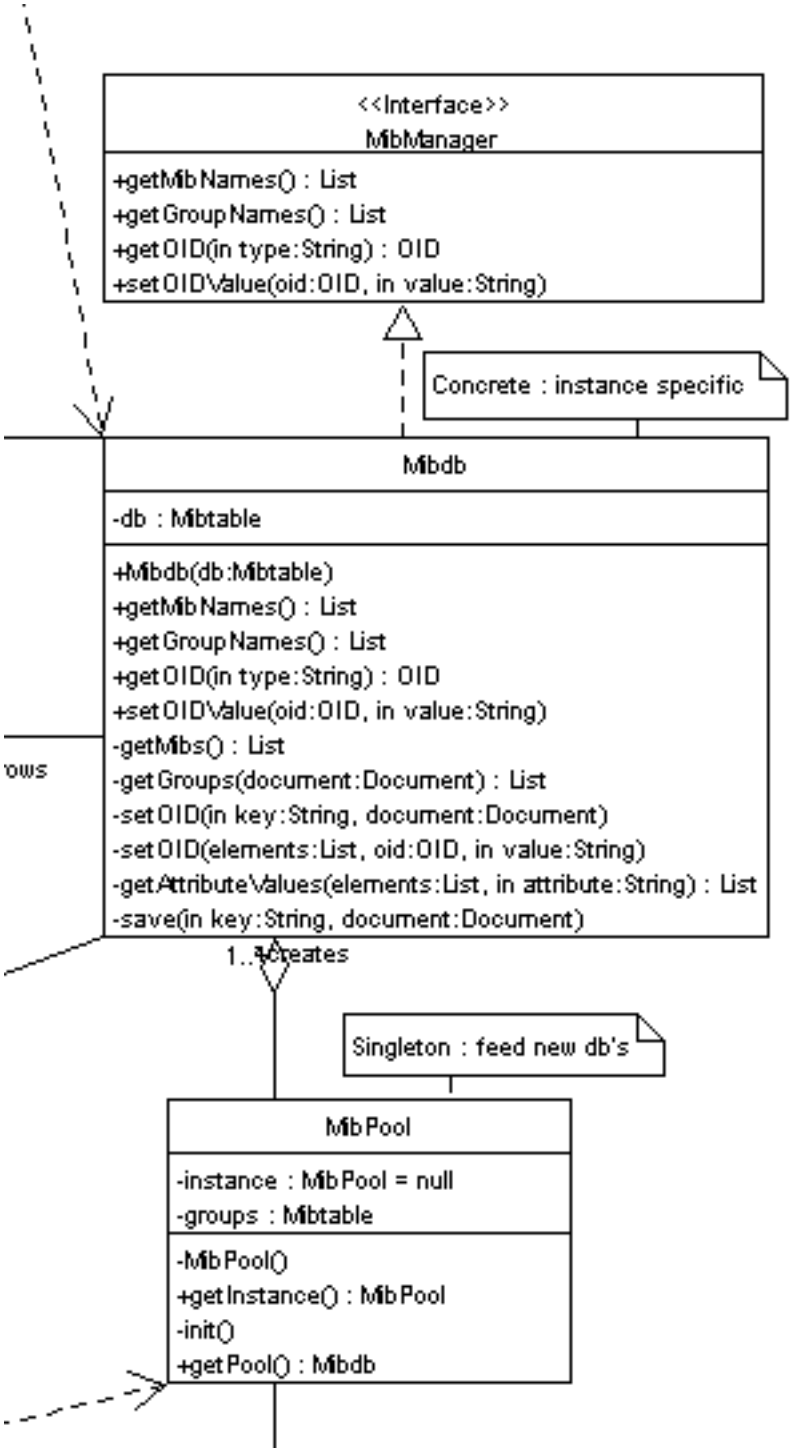
```
/** DTD attribute value */
public static final String ATTRIBUTE_OID = "oid";

/** DTD attribute value */
public static final String ATTRIBUTE_TYPE = "type";

/** DTD attribute value */
public static final String ATTRIBUTE_SYSTEM = "system";

/** DTD attribute value */
public static final String ATTRIBUTE_SYNTAX = "syntax";

/** DTD attribute value */
public static final String ATTRIBUTE_SIZE = "size";

/** DTD attribute value */
public static final String ATTRIBUTE_ACCESS = "access";

/** DTD attribute value */
public static final String ATTRIBUTE_STATUS = "status";
```



UML Diagram for the MibPool, MibTable and the MibPoolFactory.



Sequence Diagram for NTCIP GUI using the Protocol Handler to update the status of the DMS Device.



UML Diagram for the MibManager interface, the Mibdb working object and the Singleton MibPool.

NTCIP DMS Integration into CHART2

The CHART2 system provides a series of strong interfaces which carry the properties of a DMS. The NTCIP DMS only needs to implement and extend these objects to provide the required CHART2 behavior of the DMS.

A standard specific DMS in the CHART2 system must adhere to the DMS and CHART2DMS interfaces provided by the CHART2 system. These are discussed in detail in the High Level Design documentation for CHART2. The CHART2DMS interface requires the DMS to carry the CHART2DMSStatus object which contains information on the current status of the DMS. The CHART2DMSStatus contains the publicly accessible class member variables;

```
m_currentMessage
m_performingPixelTest
m_commMode
m_opStatus
m_shortErrorStatus
m_statusChangeTime
m_controllingOpCenter
```

Where m_currentMessage is a DMSMessage object, m_performingPixelTest is a boolean value, m_commMode is a CommunicationMode object, m_opStatus is an OperationalManagement object, m_shortErrorStatus is an int, m_statusChangeTime is an int and m_controllingOpCenter is an OpCenterInfo object. These objects predominantly describe the Sign's external status as opposed to the signs Mib state. The DMSMessage is described through Mib's, however the CHART2 DMSMessage object contains more information about the Message, such as it's Multi-Message format and ASCII representation. The NTCIPDMSStatus object will extend this class and contains the information specific to the NTCIP DMS's status. In this case, this object would update the Mibdb which is part of the NTCIPDMS description. The interfaces to manipulate the Mibdb through the NTCIPDMSStatus object are declared to the ORB through the IDL methods in NTCIPDMSDefs which compile to the CHART2.DMSControl.NTCIPDMSDefs package.

The DMSConfiguration object contains more detail on the DMS's properties such as;

```
m_name
m_deviceLocation
m_dmsSignType
m_signMetrics
m_pages
m_dmsTimeCommLoss
m_dmsBeaconType
m_defaultJustificationLine
m_defaultPageOnTime
m_defaultPageOffTime
```

The CHART2 system also contains in the CHART2.DMSProtocols package a DMSDeviceStatus object which contains information on the beacon state, the Multi Message and the short Error Status of the DMS. This is the object returned by the Protocol Handler's getStatus() method.

The GUI component for the NTCIP DMS is contained in the CHART2.GUIDMSModule.NTCIP package with the GUINTCIP object extending the GUIDMS object.

Resources

- CHART2 : http://www.chart.state.md.us/
- Edwards and Kelcey Technology : http://www.ekcorp.com/
- NEMA : http://www.nema.org/
- NTCIP : http://www.ntcip.org/
- Object Management Group : http://www.omg.org/

References

- Edwards and Kelcey Report Subtask 1 : NTCIP Compliance Survey and Driver Development. 2001.
- Edwards and Kelcey Report Task 23 : NTCIP implementation of PMPP in CHART2. 2001.
- Edwards and Kelcey Report Task 23 : NTCIP Mib's XML Storage Mechanism. 2001.
- MSHA Report : Performance Evaluation of CHART, An Incident Management Program. 1997.
- MSHA Report : CHARTII Release I, Build 2 High Level Design.
- MSHA Report : CHARTII Release I Build 2 - GUI Detail Design.
- MSHA Report : CHARTII Release I, Build 2 - Field Management Station detailed Design.
- MSHA Report : CHARTII Release I, Build 2 - Field Management Station High Level Design.
- MSHA Report : CHARTII Release I, Build 2A - High Level Design.
- MSHA Report : CHARTII Release I, Build 2A - Detailed Design.

Glossary

- CORBA : Common Object Request Broker Architecture.
- DMS : Dynamic Message Sign.
- DTD : Document Type Definition.
- FMS : Field Management Station.
- GUI : Graphical User Interface.
- HDLC : High-level Data Link Control.
- IDL : Interface Definition Language.
- ISDN : Integrated Services Digital Network.

- MIB : Management Information Base.
- NEMA : National Electrical Manufacturers Association.
- NTCIP : National Transportation Communications for ITS Protocol.
- OID : Object Id.
- ORB : Object Request Broker.
- PDU : Protocol Data Unit.
- PMPP : Point to Multi Point Protocol.
- POA : Portable Object Adapter.
- SNMP : Simple Network Management Protocol.
- VMS : Variable Message Sign.
- WAN : Wide Area Network.
- XML : Extensible Mark-up Language.